TP Qualité de code - Sonar

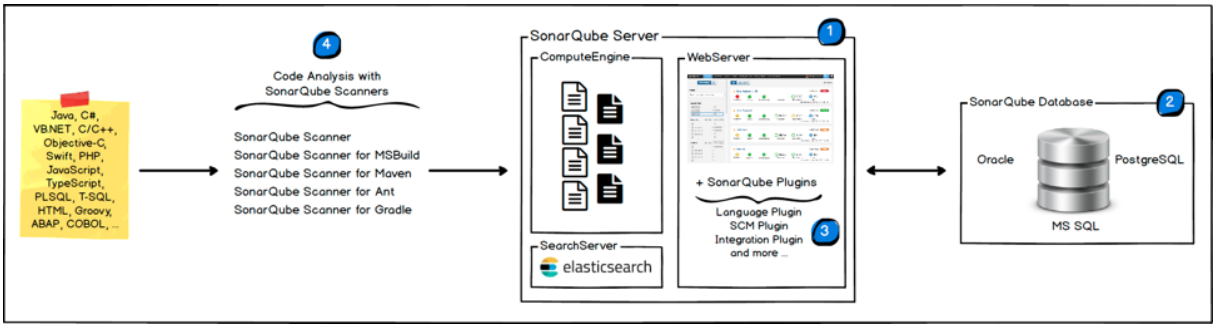
Ce TP est orienté sur le “Software Quality” et l’utilisation d’un des outils disponibles pour le mesurer : SonarQube Il existe de nombreux autres outils répertoriés sur cette page Wikipédia : List of tools for static code analysis.

Comme d’autres domaines de la qualité, la qualité logicielle est encadrée par un ensemble de normes. Nous nous concentrons ici sur la norme ISO/CEI 9126 (redéfinie par la norme ISO/CEI 25010). Nous vous recommandons de lire cette introduction aux normes 9126 et 25010 pour vous familiariser avec ces concepts : Les indicateurs de la qualité de la norme ISO 9126.

SonarQube (outil d’analyse statique de code) utilise le modèle de qualité SQALE, conforme à la norme ISO 9126. Il présente en plus de nombreux avantages (opensource, multilanguage, etc.) celui d’être – relativement – aisé à utiliser.

1 Architecture SonarQube/SonarScanners

La plateforme SonarQube se compose de 4 éléments :



1. Un serveur SonarQube démarre 3 processus principaux :
   - Serveur Web pour les développeurs et les gestionnaires pour parcourir les vues résumant la qualité des projets et configurer l’instance de SonarQube
   - Serveur de recherche basé sur Elasticsearch pour sauvegarder les recherches de l’interface utilisateur
   - Un serveur dédié à l’analyse de code (Compute Engine) pour faire puis traiter les rapports d’analyse de code et ensuite les enregistrer dans la base de données SonarQube
2. Une base de données SonarQube pour stocker :
   - la configuration de l’instance de SonarQube (sécurité, paramètres des plugins, etc.)

— les instantanés de qualité des projets, des vues, etc.

3. Plusieurs plugins SonarQube installés sur le serveur, incluant éventuellement des plugins de gestion des langues, de versionnement, d'intégration, d'authentification et de gouvernance (à paramétrer dans la fenêtre des plugins)
4. Un ou plusieurs SonarScanners fonctionnant sur les serveurs d'intégration continue ou de compilation pour analyser les projets.

Nous pouvons utiliser directement Sonar via *Sonar Cloud* et son intégration sous forme d'action GitHub 3 ou en local sur vos machines avec *SonarQube* et *SonarScanner*. Ici nous allons travailler directement avec *Sonar Cloud*.

2 Mise en place d'un projet avec GitHub

Commencez par vous créer un compte sur http://github.com et créez un premier repository. Sur votre dépôt, créez à minima un fichier `ReadMe.md` et clonez votre dépôt en local.

Nous allons coder un Tic Tac Toe avec interface texte et les classes de test appropriées. Vous pouvez si vous le souhaitez faire un fork d'un projet déjà existant.

Vous pouvez mettre un système de vérification à la volée de la qualité du code sur VS Code en activant SonarLint. Cela ouvrira dans votre éditeur une fenêtre avec un ensemble de problèmes à résoudre pour augmenter la qualité de votre code.

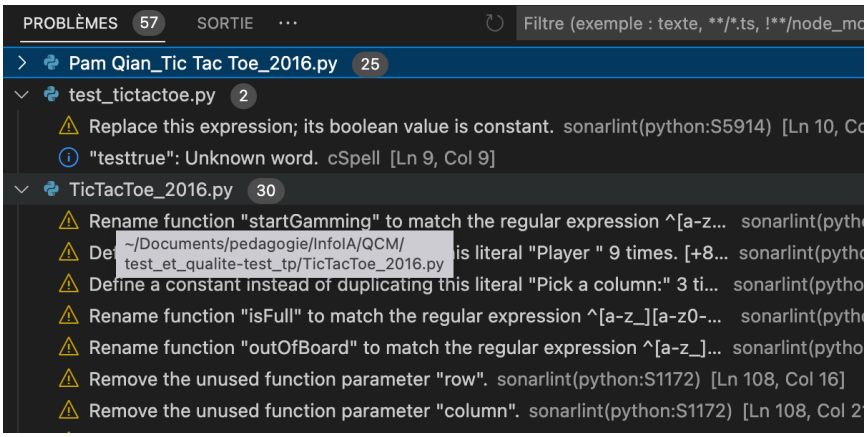


FIGURE 1 – Exemple de problèmes détectés par SonarLint dans VSCode

3 Sonar Cloud

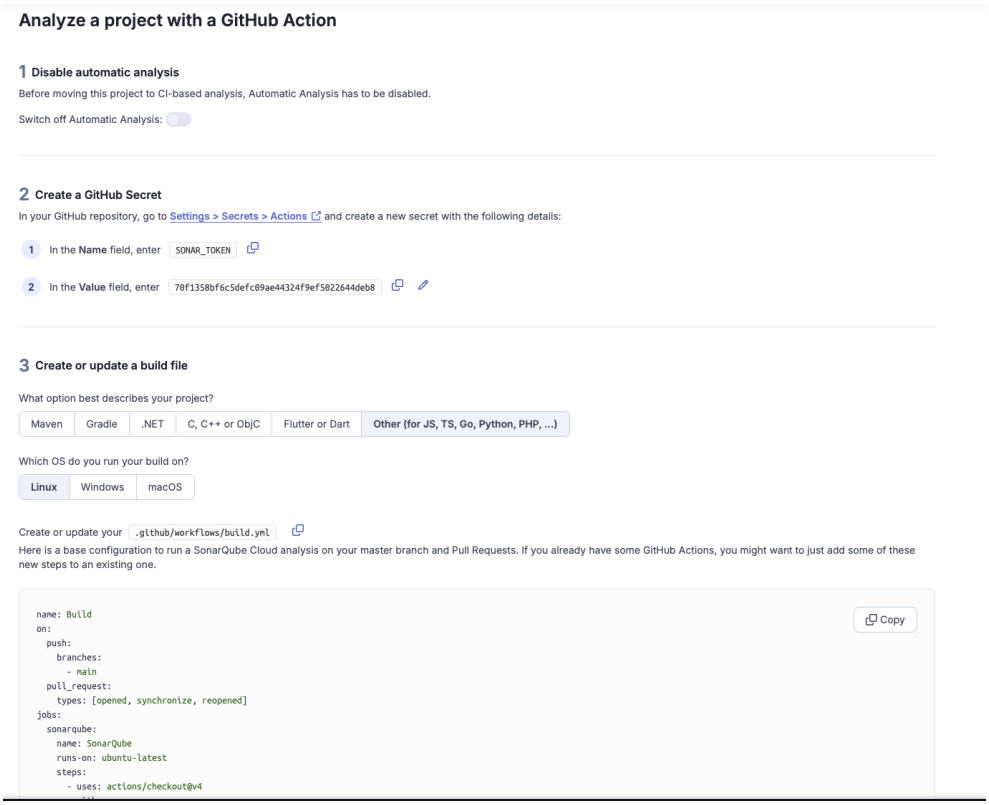
Sonar cloud est gratuit pour les projets opensource `https://www.sonarsource.com/products/sonarcloud/signup/`.

Vous pouvez suivre ce tutoriel pour que vos projets publics soient associés à SonarCloud `https://docs.sonarcloud.io/getting-started/github/`.

Une fois votre projet déclaré sur SonarCloud, déclarez l'analyse de projet comme une GitHub Action sur SonarCloud. Pour configurer cet analyse vous devez suivre un tutoriel qui est accessible en allant à la page principale de votre projet sur SonarCloud et en choisissant, dans le menu à gauche, `Administration > Analysis Method` [1].

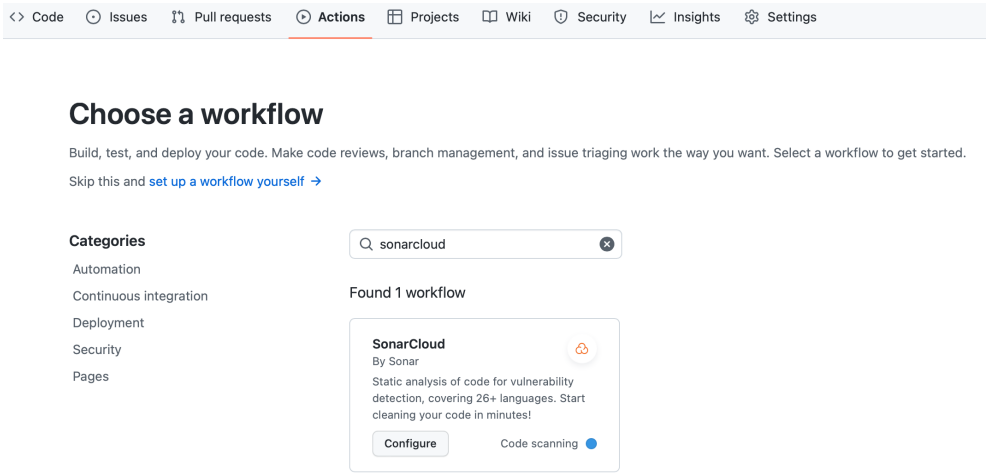
Le tutoriel vous indiquera les étapes précises pour configurer l'analyse, en résumé les étapes de base sont :

1. Créez un `GitHub secret` pour votre projet et déclarez le sur votre projet GitHub (un `New repository secret`)
2. Créez un fichier build en choisissant "Other" et "Linux". Le fichier proposé `.github/workflows/build.yml` sera ensuite à copier dans votre repository github.



3. Allez ensuite dans le menu Action de votre projet GitHub et créez un nouveau workflow SonarCloud.

1. `https://docs.sonarcloud.io/advanced-setup/ci-based-analysis/github-actions-for-sonarcloud/`



4. Créez le fichier `.github/workflows/build.yml` comme proposé dans le tutoriel sonarcloud et copier/coller le fichier `yml` dans le fichier action de github que vous êtes en train de créer.
5. Enfin, créez un fichier `sonar-project.properties` dans la racine du projet sur github avec le contenu proposé par le tutoriel sonarCloud. Nous pourrons compléter ce fichier pour personnaliser l'analyse Sonar du projet.

Remarque : sur SonarCloud, dans votre projet, le menu `Administration` > `General Setting` permet aussi de compléter les paramètres de l'analyse sonar.

Une fois le fichier créé, l'analyse se lancera et vous verrez le résultat dans SonarCloud. L'analyse sera lancée à chaque *push* sur github.

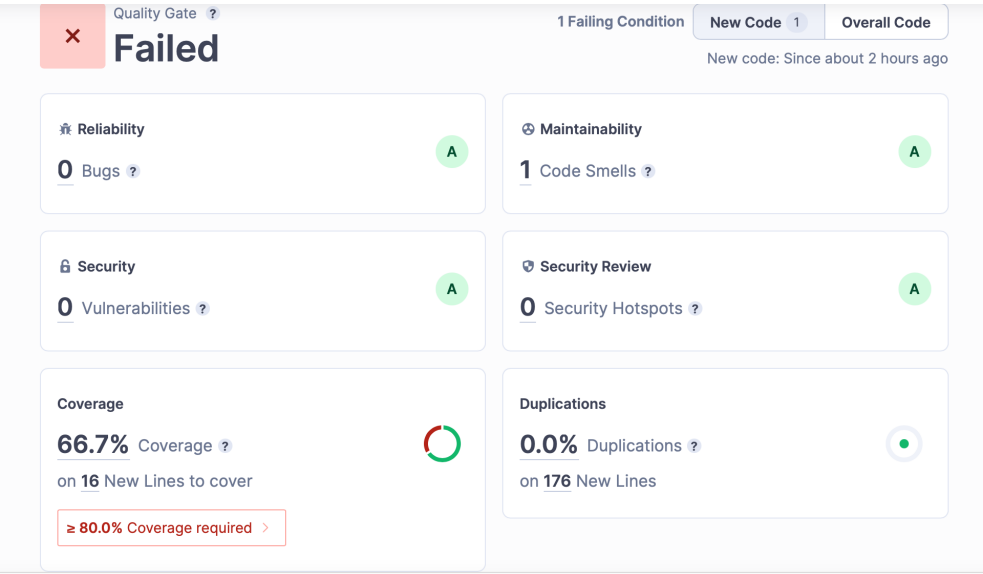
4 Ajout d'un outil de « couverture de tests »

Cette analyse permet d'ajouter une mesure de la couverture de vos tests à votre projet. Les détails de la mise en place de la mesure de la couverture dans votre processus de construction dépendent des outils que vous utilisez. Ici nous allons utiliser, comme indiqué dans le tutoriel de SonarCloud [2] :

— Tox, pour configurer les tests
— Pytest, pour exécuter les tests
— Coverage, (l'outil Coverage.py,) pour mesurer la couverture du code, et
— GitHub Actions, pour effectuer la construction.

Dans le processus d'automatisation, les tests et la mesure de la couverture de ces tests doivent être lancés au préalable de l'analyse sonar. Cette analyse de couverture de test va générer un fichier de rapport (en xml) qui sera ensuite lu par sonar pour afficher un taux de couverture comme dans la figure suivante.

2. `https://docs.sonarcloud.io/enriching/test-coverage/python-test-coverage/`



4.1 Création d'un fichier tox.ini

Dans la racine de votre projet github créez un fichier tox.ini :

```
[tox]
envlist = py39
skipsdist = True

[testenv]
deps =
pytest
pytest-cov
commands = pytest --cov=my_project --cov-report=xml --cov-config=tox.ini --cov-branch
```

Remarque : `source = .` n'est valable que si le dossier où se trouve `tox.ini` est le dossier contenant les sources python du projet à analyser. Ce chemin d'accès sera à modifier selon votre projet.

4.2 Modification du build.yml

Complétez votre fichier build.yml comme indiqué ci-dessous :

```
name: Build
on:
push:
branches:
- main
pull_request:
types: [opened, synchronize, reopened]
jobs:
sonarqube:
name: SonarQube
```

```
runs-on: ubuntu-latest
steps:
- uses: actions/checkout@v4
   with:
fetch-depth: 0  # Shallow clones should be disabled for a better relevancy of analysis
- name: Setup Python
   uses: actions/setup-python@v2
   with:
python-version: ${{ matrix.python }}
- name: Install tox and any other packages
   run: pip install tox
- name: Run tox
   run: tox -e py
- name: SonarQube Scan
   uses: SonarSource/sonarqube-scan-action@v4
   env:
SONAR_TOKEN: ${{ secrets.SONAR_TOKEN }}
```

4.3 Modification de sonar-project.properties

Ajoutez la ligne :

```
sonar.python.coverage.reportPaths=coverage.xml
```

à votre fichier `sonar-project.properties`

5 Correction du code

Complétez petit à petit votre Tic Tac Toe et les tests associés et observez les différentes métriques de votre code sur SonarCloud.